低功率低密度同位元查核碼解碼器設計

湯雲欽、陳威仁、顏偉光、陳怡孜

摘 要

本文提出了一個高傳輸速度的低密度同位查核碼解碼器之硬體設計電路，使用矩陣大小爲 512×1024、行權重及列權重分別爲 3、6 之規則隨機查核矩陣，在硬體架構上分成四個單元，分別爲：變數點單元、查核點單元、記憶體單元及配置單元。記憶單元利用了雙路徑暫存器檔案加上暫存器架構而成，並經由適當的規劃大幅降低了暫存器檔案所需要的個數及面積；查核點單元以最小-總和演算法來做爲硬體設計原則。

在傳輸速度方面，由於傳統的解碼方式在對一段編碼進行解碼時，變數點單元及查核點單元有一半的時間沒有作用，利用適當的編排讓這一半的空閒的時間來進行解碼，如此可讓傳輸速度提升至原本設計的兩倍，以大幅降低功率。

本設計以台積電 0.18 微米互補式場效電晶體技術合成，在頻率爲 50MHz、解碼次數爲 8 次時，傳輸速度可達到 1.47Gbps，面積爲 3.6 mm^2，功率消耗爲 297 mW。

關鍵詞：低密度同位查核碼、超大型積體電路、分割轉移矩陣。

湯雲欽：修平技術學院電子工程系講師
陳威仁：修平技術學院電子工程系講師
顏偉光：中興大學電機所學生
陳怡孜：台中教育大學特殊教育研究所學生
投稿日期：981027 接受刊登日期：981221

Design of Low Power Low Density Parity Check Code Decoder

Yun-Ching Tang, Wei-Ren Chen,
Wei-Kuang Yan, Yi-Tzu Chen

Abstract

In this paper, a high throughput decoder for Low Density Parity Check Code is presented. The (512, 1024) check matrix is a regular matrix whose column weight and row weight are 3 and 6, respectively. The four units include a Variable Node Unit (VNU), a Check Node Unit (CNU), a Memory Unit and a Distributor. The Memory Unit is composed of Artisan 2-Port Register Files and registers. The size of the Register Files was greatly reduced with appropriate arrangement. The Min-sum algorithm was applied in CNU.

During the decoding process of the traditional decoding method, CNU and VNU operations are active alternatively in every decoding iteration. To increase the throughput, these idled CNU and VNU circuit blocks can be utilized more efficiently by using Partition-and-Shift method. Therefore, the throughout is increased to almost two times, and power dissipation can be reduced significantly, too.

The design was synthesized using TSMC 0.18 μm CMOS technology. It can achieve 1.47 Gbps throughput with 8 iterations in cell area of 3.6 mm^2. The power dissipation is 297 mW at clock frequency of 50MHz.

Keywords: LDPC, VLSI, Partition-and-Shift matrix.

Yun-Ching Tang, Instructor, Department of Electronic Engineering, HIT.
Wei-Ren Chen, Instructor, Department of Electronic Engineering, HIT.
Wei-Kuang Yen, Student, Department of Electrical Engineering, NCHU.
Yi-Tzu Chen, Student, Department of special education, NTCU.

壹、前言

在數位通訊系統中，通道編碼器將冗餘位元加入原本的訊息，通道解碼器利用這些冗餘位元來偵測或更正接收到的資訊，由於多了這樣的機制使得通道雜訊對於所接收的資料正確性的影響力降低。

一般而言，錯誤控制碼可分成兩種：區段碼(Block Codes)及迴旋碼(Convolutional Codes)，其差異在於編碼器內是否存在著「記憶」，也就是說，區段碼的當次編碼產生結果是由當次進入編碼器的訊息決定，而迴旋碼則不同，其編碼的產生與先前進入編碼器的訊息相關。

一個(n, k)區段碼代表原始資訊長度爲 k bits，經過編碼後成爲 n bits，也就是說冗餘位元有(n-k) bits，且(k/n)稱爲編碼速率(Code Rate)。一般而言，區段碼存在著產生矩陣 G 及查核矩陣 H，訊息向量通過產生矩陣的運算後即成爲編碼，此編碼與查核矩陣運算結果應該爲 0，解碼器即是利用查核矩陣來判斷接收到的資訊是否有錯誤甚至將之更正回來。

一個(n, k, m)的迴旋碼代表若輸入爲爲 k bits，則經過編碼後成爲 n bits(不含初始狀態)，編碼器內存 m 個「記憶」，即輸出的資訊最多會與先前第 m,m-1,…,2,1 個輸入資訊有關。在計算速率方面，「編碼器」的速率是(k/n)，而編碼速率必須再加上 m 個記憶，假若進入編碼器的訊息向量長度爲 L，則編碼率爲 $\frac{k}{n}\frac{L}{L+M}$，一般而言，$L \quad M$，所以編碼率大約爲(k/n)

低密度同位元查核碼(Low Density Parity Check Code，簡稱 LDPC Code，以下稱 LDPC Code)[2]是線性區段碼的一種，其查核矩陣中”1”的個數相對於整個矩陣而言非常的少，因此給予「低密度」之稱呼。

LDPC Code 創造於 VLSI 技術尚未成熟的 1962 年，近年來因爲 VLSI 技術不斷提升，LDPC Code 開始受到重視，其高度平行化運算的特性使得傳輸速大幅提升，再加上夠大的區塊長度下可使解碼效能接近審農極限[1] [3]，這些優良特性使得越來越多學者對此錯誤更正碼產生興趣。

貳、低密度同位元查核碼

LDPC Code 解碼器的硬體架構可分爲四個單元，分別是：變數點單元(Variable Node Unit)、查核點單元(Check Node Unit)、記憶體單元(Memory Unit)及配置單元(Distributor)，顧名思義，變數點\查核點單元的功能是在做變數點\查核點的運算，記憶單元則是用來儲存變數點\查核點單元運算過後的更新資

[1] 審農極限(Shannon Limit)亦稱 Shannon Capacity，在此可解釋爲在某通道上的最佳解碼效能。

訊，配置單元是用來連結記憶體單元及變數點\查核點單元，圖 1 是上述四個單元的關係示意圖，配置單元將解調器傳送的資料(LLR)儲存至變數暫存器，同時也配置到變數點單元，變數點單元收到資料並運算過後輸出，配置單元再將變數點單元的輸出更新資訊分配到記憶體單元，此步驟稱爲「初始化」；待查核點運作時，配置單元給予記憶單元適當的「位址」值使之輸出正確所需的資料，再將此資料送進查核點單元運算，運算後的更新資訊經查核點單元輸出，配置單元再將此資訊存回記憶體單元，此步驟稱爲「查核」；配置單元再給予記憶體單元及 LLR Reg.適當的「位址」值使之輸出正確所需的資料並分配至變數點單元，變數點單元收到資訊後進行 Hard Decision，並將更新的資訊經由分布單元傳送回記憶體單元，此步驟稱爲「蒐集」；一次的查核加一次的蒐集稱爲一次解碼。

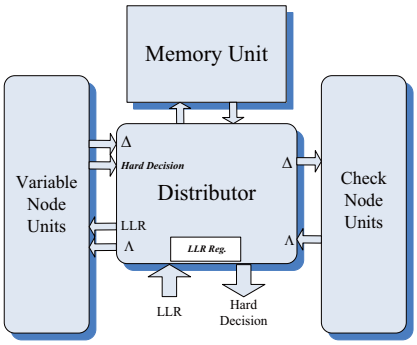

圖 1、LDPC Code 解碼器之硬體架構示意圖

如圖 2 所示，*H* 是 LDPC Code 的查核矩陣，大小爲 512×1024，本論文所設計的解碼器屬於部份平行式解碼器，即變數點運算及查核點運算皆不是一次做完，將行及列都平均分割成 4 部分，每次只做其中一部份，因此，每一次的變數點運算或查核點運算皆要進行四遍才能完成，這樣的分割雖造成解碼時間拖長，大也大幅減少所需要的變數點單元及查核點單元數量，如圖中所示，切割過後，變數點運算每一部分有 256 行，因此圖 4.1 的變數點單元含有 256 個變數點運算；同理，查核點單元含有 128 個查核點運算。另一方面，也因爲是部分平行式運算的關係，所以需要記憶體單元來儲存已完成運算部分的資訊。

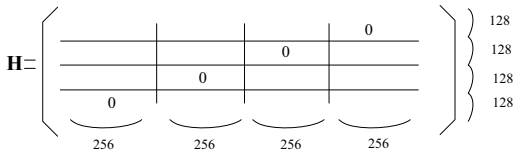

圖 2、查核矩陣 H

由於其行權重固定爲 3，列權重固定爲 6，因此就最基本的硬體設計，變數點單元應有 4 個輸入(包含接收解調器資訊的 *LLR* 埠)及 4 個輸出(包含 *Hard Decision*)，輸入部分皆是 5 bits，輸出部分有 3 個是 5 bits，1 個(*Hard Decision*)是 1bit；而查核點單元應該有 6 個輸入及 6 個輸出，皆爲 5 bits。

圖 3 是數字系統的變化圖，解調器傳送資訊(LLR)至 LDPC Code 解碼器，其使用的數字系統是 *Sign Magnitude*(簡稱 *SM*)，進入變數點後先轉爲 *2's complement*(簡稱 *TC*)後再做變數點運算，運算結果再轉回 *SM* 輸出且儲存回記憶體單元，而查核點的運算過程皆是使用 *SM* 數字系統。也就是說，本解碼器設計在資料表示方面，數字系統是以 *SM* 爲主，僅有在進行變數點運算時需要做數字系統轉換。

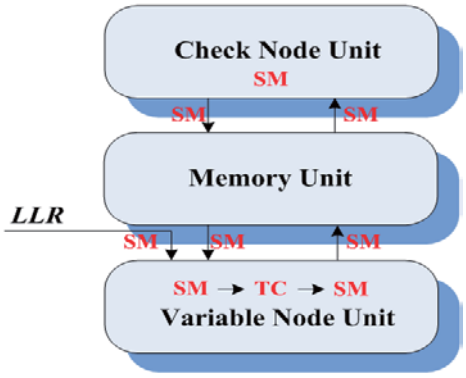

圖 3、LDPC Code 解碼器之數字系統轉換圖

系統在「初始化」階段時，變數點單元的 LLR 輸入埠接收解調器所傳送的資訊，同時此資訊也儲存到 LLR Reg.，而在之後的解碼動作，LLR 輸入埠僅讀取 LLR Reg，此部分的設計如圖 4；由於每個資訊有 5 bits，編碼資訊長度是 1024 分四次由解調器輸入，因此所需要的 LLR Reg.大小爲 4 × 1280，但如同前段文章所述，此大小的 Register File 必須由 10 個大小是 4 × 125 的 Register File 與 1 個 4 × 30 所組成。

A 256	E 256	J 256	N 0
B 256	F 256	K 0	O 256
C 256	G 0	L 256	P 256
D 0	I 256	M 256	Q 256

圖 4、查核矩陣 H 的 1 分布數量

另一方面，系統在初始化階段時，僅是要將 LLR 值分布到記憶體單元內，因此此時的變數點單元的三個輸入 Λ_1~Λ_3 都先設爲 0，而在之後的解碼動作時，Λ_1~Λ_3 會依序接上第一行至第四行內的資訊所儲存在的 Register File，也就是說，在解碼階段，變數點單元的三個輸入埠 Λ_1~Λ_3 都最多會接上 4 個不同的 Register File，再加上初始化階段的來源，這三個輸入埠中，每一個最多會有 5 種不同的輸入來源，因此，變數點單元中輸入埠 Λ_1~Λ_3 的來源分布有 5 種狀態，另外，***VNOEN*** 訊號則是表示系統是否已進行至最後一次解碼，最後一次解碼時狀態設爲 LOW；在查核點單元方面，由於此單元不參與系統初始化階段，因此 Δ_1 至 Δ_6 僅在解碼階段時依序接上圖 4 中第一列至第四列內的資訊所儲存在的 Register File，也就是說，查核

點單元的六個輸入埠中，每一個最多會有 4 種不同的輸入來源，因此，查核點單元中輸入埠 Δ_1 至 Δ_6 的來源分布有 4 種狀態。其詳細的設計如圖 5。

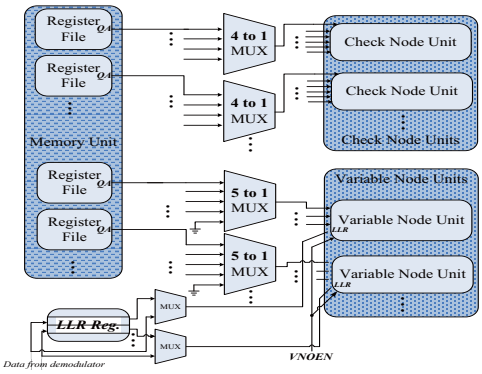

圖 5、單路徑配置單元處理運算單元的輸入資訊設計圖

在運算單元的更新值儲存的設計方面，在系統解碼階段時，運算單元的更新值所欲儲存的目的 Register File 即是原始輸入值的來源 Register File，再者，在系統初始化階段時， LLR 值的分布目的 Register File 等同於在解碼階段時更新值的分布，因此不管是變數點或查核點的輸出分佈皆有 4 個狀態。若改以 Register File 接收資料的觀點，在儲存各運算單元的輸出值方面，每個 Register File 最多可從每個運算單元得到 4 種不同的來源，因此共有八種來源，也可以說，Register File 輸入埠 DB 的來源分布有 8 種狀態，此部分的詳細設計如圖 6。另一方面，變數點單元輸出埠 A 的 *MSB* 會與 ***VNOEN*** 訊號做 OR 運算，待最後一次解碼，***VNOEN*** 設爲 LOW，系統則會輸出 A 的 MSB，即 Hard Decision。

在更進一步的探討變數點單元及查核點單元存取資料的次序，如圖 4，變數點單元存取資料是由左至右：行 1、行 2、行 3、行 4，查核點單元是由上至下 ：列 1、列 2、列 3、列 4，由 Register File 取出的資料經運算後，更新值會於下一週期才儲存回 Register File，因此在變數點運算進到查核點運算，或是查核點運算進到變數點運算時會發生配置單元要針對同一塊 Register File 的同一個位址在同一週期進行存取，例如：進行列 4(D、I、M、Q)的查核點運算後，在下一週期會將運算結果存回列 4 的 Register File 分布，同時間也要讀取行 1(A、B、C、D)的資料作變數點運算。

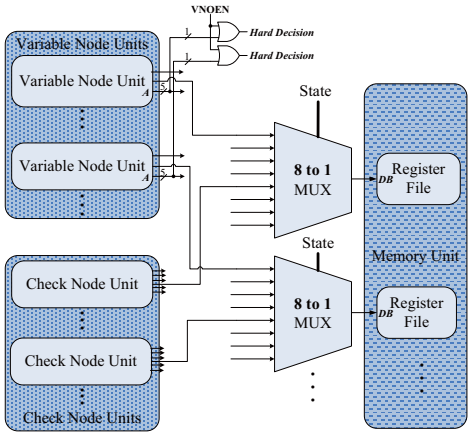

圖 6、單路徑配置單元處理運算單元的輸出資訊設計圖

圖 7 訊號狀態的時序圖，方框內的數字代表該訊號在當週期屬於哪一個狀態；將變數點單元輸入埠 Λ_1~Λ_3 接 0 時設爲狀態 4，讀取行 1 至行 4 所對應的 Register File 分別屬於狀態 0 至狀態 3，LLR 輸入埠直接讀取解調器數值時爲狀態 0，讀取 LLR Reg.時爲狀態 1，LLR Reg AA/AB 上的數值代表在該週期被讀取或欲儲存的資料位於那一個位址；查核點單元輸入埠 Δ_1~Δ_9 讀取列 1 至列 4 所對應的 Register File 分別屬於狀態 0 至狀態 3，Register File 預計儲存行 1 至行 4 的運算結果(變數點運算)分別屬於狀態 0 至狀態 3，預計儲存列 1 至列 4 所對應的的運算結果(查核點運算)Register File 分別屬於狀態 4 至狀態 7。方框內劃 X 即代表該訊號所屬於的單元在該週期沒有運作。如圖所示始化需要四個週期，一次解碼需要八個週期，系統在進行到第 12 週期的狀態後會再回到第 5 週期的狀態進行下一次解碼。圖 8 解碼器進行至最後解碼的狀態時序圖，在最後一次解碼時，變數點是進行 Hard Decision，因此只需要將編碼中屬於訊息的部分解碼即可，所以只將行 1 至行 3 所對應的 Register File 做運算，也就是說，變數點單元在最後一次解碼時，輸入埠 Λ_1 至 Λ_3 的資料連結分佈只經歷狀態 0 至狀態 2。

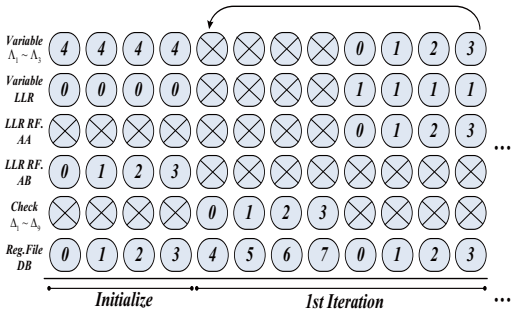

圖 7、單路徑 LDPC Code 解碼器狀態變化時序圖(前段)

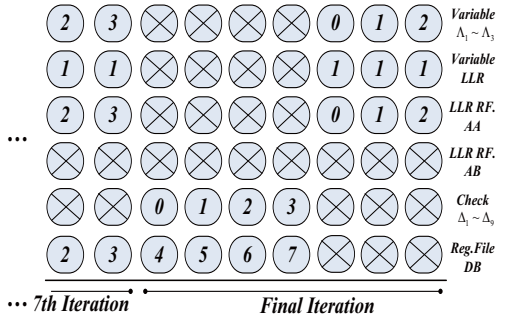

圖 8、單路徑 LDPC Code 解碼器狀態變化時序圖(後段)

參、低功率解碼器設計

觀察圖 7 可知，扣除初始化階段，在整個解碼過程中，變數點單元及查核點的使用率皆只有 50%，本節提出的方法，不僅使得解碼階段變數點單元及查核點單元的使用率可達到 100%，並且提高了系統整體的解碼速度。

就整體硬體而言，可以讓變數點單元與查核點單元同時運作而不會額外的增加硬體，所以爲了讓變數點單元與

查核點單元可以同時運作，所以提出如圖 4 的查核矩陣 H，此查核矩陣 H 最好的好處就是，可以讓變數點單元與查核點單元在切換時提前一個週期運作。

以另一個觀點來描述新型解碼器，我們可以說：新型解碼器是變數點單元與查核點單元是在同一週期工作的，如圖 9，其中的在變數點單元與查核點單元之前各有一個多工器，由多工器來判斷說此時資料進入數點單元或查核點單元之後是否要存入記憶體單元

而我們讓變數點單元與查核點單元同時運作有一個很大的問題需要突破，就是頻率不能設太大，因爲同一週期要完成兩樣工作，假使每一區塊 1 的各數不一樣時，會使變數點單元或查核點單元完成的時間長短不一，這樣的話就會變成同樣的時間內，有些週期會提早完成工作，有些會還沒完成，所以此種架構最好搭配如圖 4 這種查核矩陣，而這種查核矩陣的特色是，全部皆由單位矩陣的平移所組成，所以每一變數點單元與查核點單元所輸入的資料長度會一樣，每一週期所需完成的工作量皆會相等，這樣一來就可以解決上述所提出的問題。

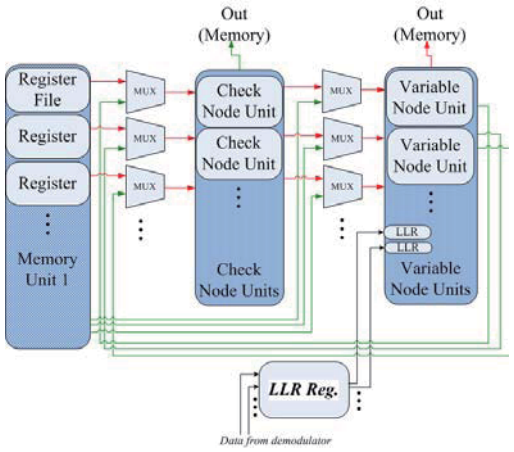

圖 9、記憶體單元的輸出資訊設計圖

圖 10 是配置單元中資料由運算單元至記憶體單元的設計示意圖，如同上節所述，變數點或查核點的輸出分佈皆有 4 個狀態，若改以接收這些更新輸出資料爲觀點，將接收變數點單元及查核點單元的資料分成兩組，即一個變點數單元需要六個四對一的多工器，而一個查核點單元需要兩個四對一的多工器。在未經過編排的情況下，在週期爲 20 ns 時，本解碼器的解碼功率爲 649mW，而面積爲 5.8 mm^2。

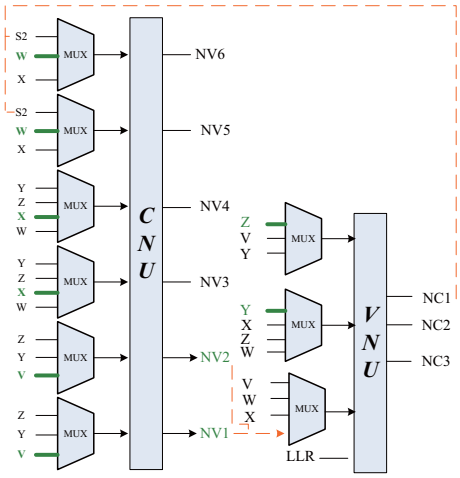

圖 10、未經過編排的配置單元處理運算單元設計圖

圖 11 是訊號狀態的時序圖，LLR 輸入埠直接讀取解調器數值時爲狀態 0，讀取 LLR Reg.時爲狀態 1，在 Initialize 之後會先做兩次的查核點單元，完成這兩次運算之後，訊號就會每 4 個週期變動一次，也就是說，除了第一次 Iteration 需要 6 個週期之外，從第二次 Iteration 開始每次都只需要 4 個週期， Register File 預計儲存行 1 至行 4 的運算結果(變數點運算)分別屬於狀態 0 至狀態 3，預計儲存列 1 與列 2 所對應的的運算結果(查核點運算)Register File 分別屬於狀態 4 至狀態 5，預計儲存列 3 行 1(變數點運算與查核點運算)所對應的的運算結果 Register File 屬於狀態 6，預計儲存列 4 行 2 所對應的的運算結果 Register File 屬於狀態 7，預計儲存列 1 行 3 所對應的的運算結果 Register File 屬於狀態 8，預計儲存列 2 行 4 所對應的的運算結果 Register File 屬於狀態 9。

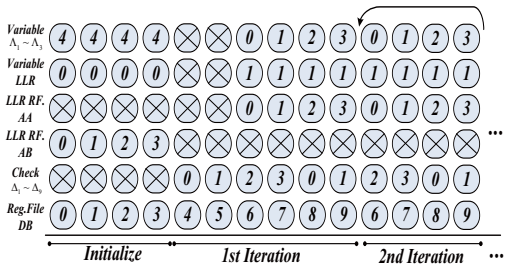

圖 11、解碼器狀態變化時序圖(前段)

圖 12 是解碼器進行至最後解碼的狀態時序圖，解碼週期減少爲 34 個週期，在週期爲 20 ns 時，本解碼器的功率爲 649mW，而解碼速度爲 753Mbps。

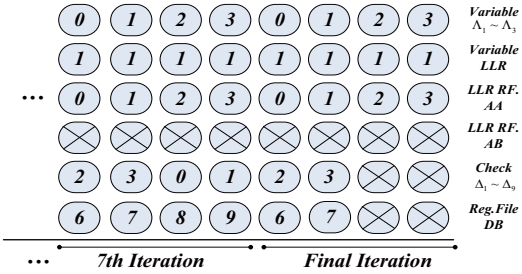

圖 12、解碼器狀態變化時序圖(後段)

在圖 13 中爲多種不同方法的時序圖，第一種爲單純的單路徑，即變數點單元與查核點單元分開動作，變數點單元做完再做查核點單元，這樣的話整個動作會拉很長，但是所使用的硬體也是最少的；而第二種爲 overlap 這種方法適用於左下角與右上角爲零矩陣的查

核矩陣，也就是在變數點單元或查核點單元切換時，同時做另一單元，這樣的話解碼速度會有所提升，但如圖所示有些時間還是沒有運作的，所以還可以有改善的空間；第三種爲雙路徑解碼，此種方法也是讓變數點單元與查核點單元同時運作，此種方法的速度非常快，但是硬體的需求量也相對的大增，它需要兩倍的記憶體來儲存資料，所以面積很大，也非常耗電；第四種是我們提出的分割轉移解碼，它的時序圖與雙路徑非常類似，而且由於查核矩陣經過設計，所以記憶體比雙路徑少，面積也比較小，也可以達到變數點單元與查核點單元同時工作的功能，相對的也會比較省電。

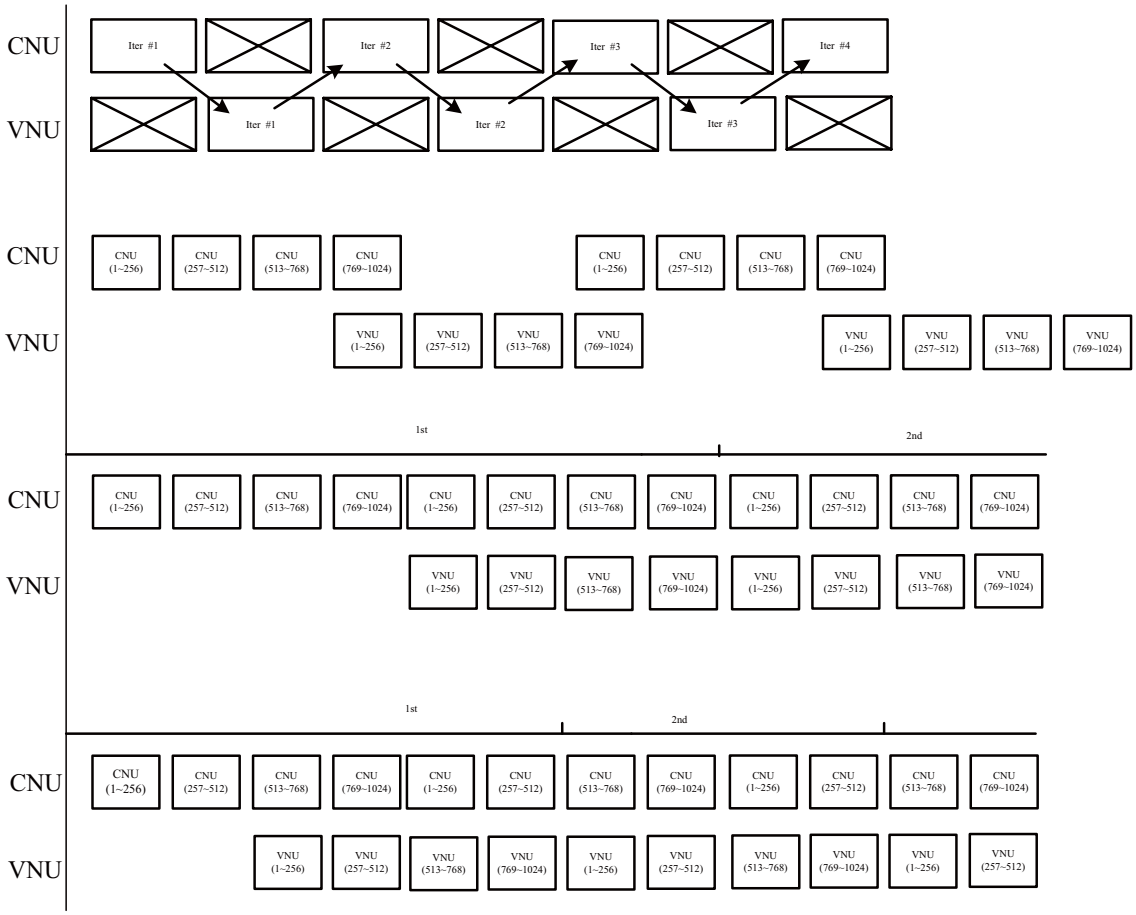

圖 13、四種方法之時序圖

爲了降低 LDPC Code 解碼器消耗的功率，另提出一個查核矩陣，此查核矩陣經過特殊的編排可以減少多工器的切換，在記體方面也使用四塊暫存器取代其中的四塊記憶體，以降低功率。

經過測試之後 LDPC Code 解碼器消耗的功率分爲兩大部分，一部分是龐大的記憶體，另一個就是在配置單元中多工器的切換，但由於記憶體在設計流程中像是一個黑盒子，只能由記憶體產生器產生，所以並不能從中去取得最佳化，在本章第二節記憶體單元中提到，要解決在這個週期把資料寫入，在下個週期就必須把資料讀出，這種「立即需使用的資料」，解決的方法有兩種，一是改成使用正反器，假如硬是要晶片中心的記憶體，就必須使用兩個時脈週期，一個時脈給寫入使用，另一個時脈給輸出使用，在第三節中就是使用這種方法，是這樣的話最多只有一半的週期可以工作，可以說非常不符合效應。

首先改善記憶體的問題，在圖 5 中有四塊記憶體在前面四塊中都只有用到兩個位址，但是由記憶體產生器產生的記憶體，最少都要四個位址，也就是說有兩個位址是沒使用的，而且在這四塊記憶體中有兩塊會有「立即需使用的資料」，所以在這節中，保留第五塊記憶體，其他的記憶體都使用正反器取代，這樣的話就可避免浪費沒有使用的記憶體，也可以避免掉只有一半週期可以工作的問題。

Z0 (13,6)	Y0 (121,80)	V2 (1,1)	0
Y1 (124,53)	Z1 (104,114)	0	V3 (1,1)
V0 (1,1)	0	X0 (13,62)	W0 (18,85)
0	V1 (1,1)	W1 (93,122)	X1 (43,53)

圖 14、經過編排的查核矩陣

如圖 14 所示經過編排的查核矩陣，VWXYZ 爲相對應那些區塊所需存入或讀取的記憶體，而後面的數字可表示成其相對應的位址，也就是說 VWXY 有 2 種位址，而 V 有 4 種位址，而此種查核矩陣的特點是在對應於 E 的記憶體，由於區塊內的 1 都是由單位矩陣所組成，也可以說是完全沒有經過平移的單位矩陣，所以在他要存入或讀取時，是不需要經過轉換就可以直接套用於變數點單元或查核點單元，這樣一來就可以節省多工器的切換。

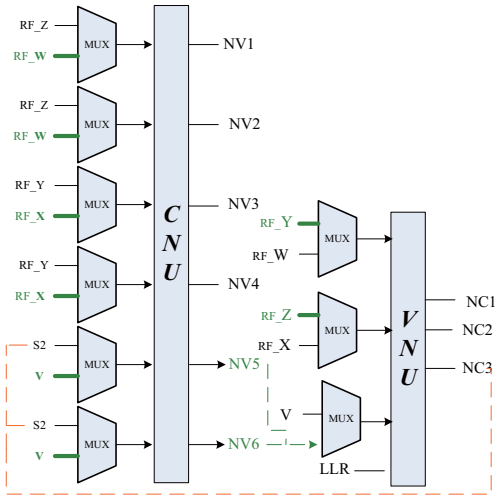

圖 15、經過編排的配置單元處理運算單元設計圖

在圖 15 中可看出，經過編排後的硬體中變點數單元前只需要 3 個二對一的多工器，而查核點單元前只有 6 個二對一的多工器，而此架構總共使用了 128 個變點數單元、256 個查核點數單元，所以節省下來的功率是非常可觀的，本解碼器的速度爲 753Mb/s，所消耗的功率降爲 297mW。

肆、比較

表 1 是本文設計各種設計的規格比較表，在傳統的方法(變點數單元與查核點單元分開運算)及單純使用記憶體及使用記憶體加暫存器下，合成後的 Total Cell Area 分別是 4.6、5.8 、3.6 及 3.9 mm^2。

表 1、結果比較表

	Traditional	Proposed (1)	Proposed (2)	
Block Length	1024			
Code Rate	1/2			
Technology	0.18-μm			
Clock frequency	50MHz			62.5 MHz
Gate counts	460k	580k	360k	390k
Data Rate	426Mb/s	753 Mb/s		919 Mb/s
Power	277mW	649mW	297mW	370mW
Throughput	0.852G	1.506G		1.838G

表 2 爲本文所提出的 LDP Code 解碼器與其他論文的效能比較，[10]爲同樣使用 coding rate 爲 1/2 條件下，本文架構大幅降低工率的消耗之外，而傳輸速率也可以有所提升； 而[11] 與[12]的製程分別爲 90-nm 與 0.16-μm，相同之處在於都是使用 coding rate 可調的方法在[11]中雖然頻率可到達 150 MHz，而本架構只有 62.5MHz 但是在傳輸數率上還有更好的表現;[13]爲 coding rate 爲 2/3 情況下，雖然傳輸速率上有比較好的表現，但是功率消耗也非常的可觀，與本架構的比較之後，面積與功率方面都做了大幅度的改善。

表 2、結果比較表

	[10]	[11]	[12]	[13]	Proposed
Block Length	1024	576~2304	660	1500	1024
Coding rate	1/2	1/2~5/6	11/15	2/3	1/2
Technology	0.16-μm	90-nm	0.13-μm	0.18-μm	0.18-μm
Clock frequency	64 MHz	150 MHz	300MHz	100 MHz	50MHz
Gate counts	1750k	970k	690k	611k	360k
Data Rate	512Mb/s	---	2.4 Gb/s	1.4Gb/s	753 Mb/s
Power	690mW	264mW	1383mW	878mW	297mW
Throughput	1.024G	0.1G	3.3G	2.24G	1.506G

伍、結論

本論文所提出的 LDPC Code 解碼器設計使用了 Artisan 2-Ports Register File 搭配暫存器來減少記憶體單元所佔的硬體面積，另一方面也減少多工器的使用，來達到低功率的效果，其設計方式是將查核矩陣上的資料加以編排，使其能在每個運算週期所利用的暫存器最少。

經合成的結果，低功率轉移分割 LDPC 解碼器的 power 爲 297mW，相比於其他設計有較小的 power，但是還有改進的空間，因爲查核矩陣不能太有規則，否則解碼的效能會很差，若要針對此問題改善的話要由基礎的查核矩陣著手，找出別種查核矩陣有著亂數的特性，但可以讓其資料進路記憶體時有規律，這樣的話就可以避免多工器的切換，讓 power 可以有下降的空間。

陸、參考文獻

[1] Simon Haykin, “Communication Systems 4th Edition,” John Wiley & Sons, Inc.

[2] R. G. Gallager, “Low density parity check codes,” IRE Trans. Inf. Theory, vol. IT-8, no. 1, pp. 21–28, Jan. 1962.

[3] C. E. Shannon, “A Mathematical Theory of Communication,” Bell System Technical Journal, vol. 27, pp. 379-426, 623-656, July, October, 1948.

[4] Thomas J. Richardson and Rudiger L. Urbanke, “Efficient Encoding of Low-Density Parity-Check Codes,” IEEE Transactions on Information Theory, vol. 47, no. 2, pp.638 – 656, Feb 2001

[5] Xiao-Yu Hu, Evangelos Eleftheriou, Dieter-Michael Arnold, and Ajay Dholakia, “Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC Codes,” IEEE Global Telecommunications Conference, pp.1036-1036E, 2001

[6] Jin Lu Membe and José M. F. Moura, “Partition-and-Shift LDPC Codes” IEEE Transactions on Magnetics, vol. 41, no. 10, October 2005

[7] Artisan Standard Library Register File Generator User Manual

[8] Shu Lin, J. Daniel, Jr. Costello, “Error Control Coding Fundamentals and Applications,” New Jersey, NJ: Prentice-Hall, 1983.

[9] Sang-Min Kim and Keshab K. Parhi, “Overlapped decoding for a class of quasi-cyclic LDPC codes,” IEEE Signal Processing Systems (SIPS

2004), pp.113-117, 2004.

[10] A. Blanksby and C. Howland, “A690-mw1-Gb/s 1024-b,rate-1/2 low-density parity-check code decoder,” IEEE J. Solid-State Circuits, vol. 37, no. 3, pp. 404–412, Mar. 2002

[11] Chih-Hao Liu, Shau-Wei Yen, Chih-Lung Chen, Hsie-Chia Chang, Chen-Yi Lee, Member, IEEE, Yar-Sun Hsu, and Shyh-Jye Jou, “An LDPC Decoder Chip Based on Self-Routing Network for IEEE 802.16e Applications,” IEEE Journal of Solid-State Circuits, vol. 43, no. 3, March 2008

[12] Ahmad Darabiha, Anthony Chan Carusone and Frank R. Kschischang “Power Reduction Techniques for LDPC Decoders”, IEEE J. Solid-State Circuits, vol. 43,no. 8,pp.1835-1845,Aug 2008

[13] 林彥志, “低功率低密度同位元查核碼解碼器設計”, 2007年碩士論文, 國立中興大學。

[14] Chien-Ching Lin, Kai-Li Lin, Hsie-Chia Chang and Chen-Yi Lee, “A 3.33Gb/s(1200,720) Low-Density Parity Check Code Decoder,” Proc. of European Solid-State Circuits Conference, pp. 211-214, 2005

[15] Sang-Min Kim; Parhi, K.K., “Overlapped decoding for a class of quasi-cyclic LDPC codes,” IEEE Signal Processing Systems, pp. 113-117, 2004.

[16] Hao Zhong and Tong Zhang, “Design of VLSI Implementation -Oriented LDPC Codes,” IEEE Semiannual Vehicular Technology Conference (VTC) , pp.670-673, Oct. 2003

[17] Jeremy Thorpe, “Design of LDPC Graphs for Hardware Implementation,” IEEE International Symp. Information Theory, pp. 483, 2002

[18] Emmanuel Boutillon, Jeff Castura, Frank R.Kschischang,“Decoder-First Code Design,” Proc. of the 2nd International Symposium on Turbo code and Related Topics, Brest, France, pp. 459-462, Sep. 2000

[19] Joachim Hagenauer, Elke Offer, and Lutz Papke, “Iterative Decoding of Binary Block and Convolutional Codes,” IEEE Transactions on Information Theory, vol. 42, no. 2, pp. 429-445, Mar. 1996